CDN (Content Delivery Network)

Suresh Podeti

8 min read · Sep 23, 2023

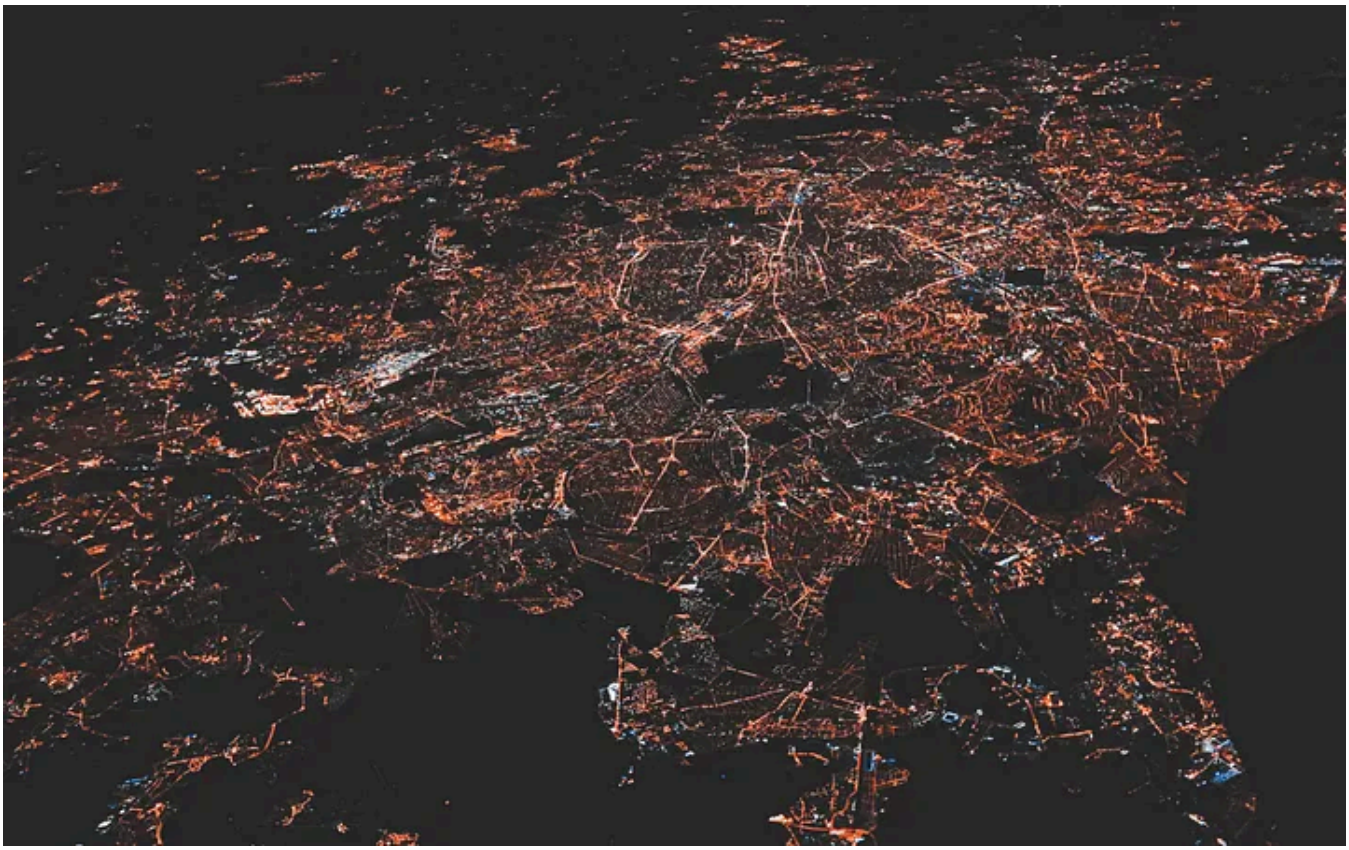

Photo by Nastya Dulhiier on Unsplash

Introduction

A CDN (Content Delivery Network) is a group of geographically distributed proxy servers. A **proxy server** is an intermediate server between a client and the origin server. The proxy servers are placed on the network edge. As the network edge is close to the end users, the placement of proxy servers helps quickly deliver the content to the end users by **reducing latency** and saving bandwidth.

A few well-known CDN providers are **Akamai, StackPath, Cloudflare, Rackspace, Amazon CloudFront, and Google Cloud CDN**

Requirements

- **Performance:** Minimising **latency** is one of the core missions of a CDN. The proposed design should have minimum possible latency.
- **Availability:** CDNs are expected to be available at all times because of their effectiveness. Availability includes protection against attacks like DDoS.
- **Scalability:** An increasing number of users will request content from CDNs. Our proposed CDN design should be able to scale horizontally as the requirements increase.

Design

CDN components

The following components comprise a CDN:

- **Origin servers:** Server where the web content originates. This server hosts the original or master copy of your website's content, including web pages, images, videos, stylesheets, scripts, and other assets.
- **Clients:** End users use various clients, like browsers, smartphones, and other devices, to request content from the CDN.
- **Distribution system:** The distribution system is responsible for distributing content to all the edge proxy servers to different CDN facilities.
- **Edge Servers / Proxy servers:** The proxy or edge proxy servers serve the content from RAM to the users. Proxy servers store hot data in RAM, though they can store cold data in SSD or hard drive as well.
- **Routing system:** The routing system directs clients to the nearest CDN facility.

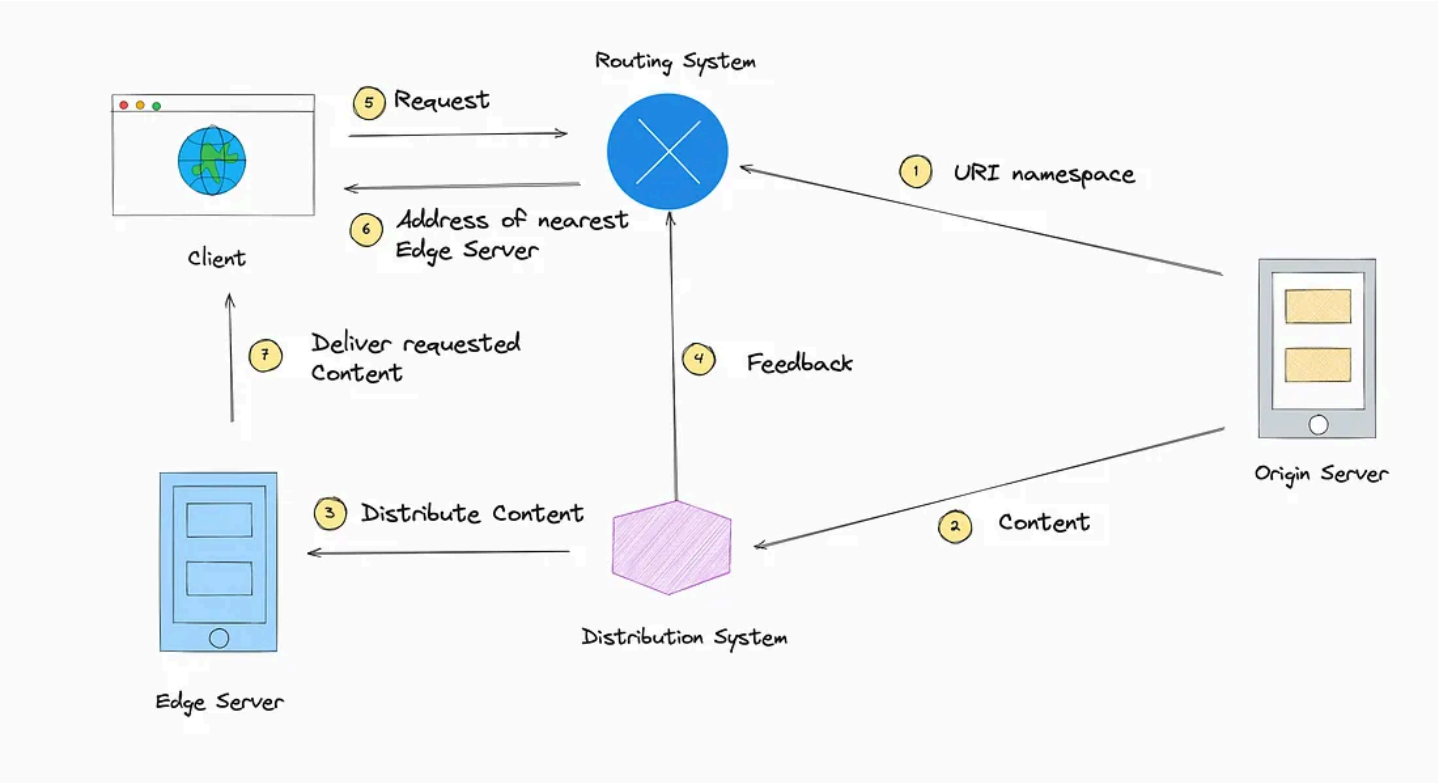

Fig 1.0: CDN components

Workflow

The workflow for the abstract design is given below:

1. The **origin servers** provide the URI namespace delegation of all objects cached in the CDN to the request routing system.
2. The origin server publishes the content to the **distribution system** responsible for data distribution across the active edge proxy servers.
3. The distribution system distributes the content among the **edge servers** / proxy servers and provides feedback to the request routing system. This feedback is helpful in optimizing the selection of the nearest proxy server for a requesting client. This feedback contains information about which content is cached on which proxy server to route traffic to relevant proxy servers.
4. The **client** requests the **routing system** for a suitable proxy server from the request routing system.
5. The request routing system returns the IP address of an appropriate proxy server.

6. The edge proxy server serves the client request. However, the request is routed to the origin servers if the content isn't available in the proxy servers. It's also possible to have a **hierarchy of proxy servers** if the content isn't found in the edge proxy servers. For such cases, the request gets forwarded to the parent proxy servers.

Content caching strategies in CDN

Identifying content to cache is important in delivering up-to-date and popular web content. To ensure timely updates, two classifications of CDNs are used to get the content from the origin servers.

Push CDN

Content gets **sent automatically to the CDN proxy servers from the origin server** in the push CDN model. The content delivery to the CDN proxy servers is the content provider's responsibility. **Push CDN is appropriate for static content** delivery, where the origin server decides which content to deliver to users using the CDN. The content is pushed to proxy servers in various locations according to the content's popularity. If the content is rapidly changing, the push model might **struggle to keep up** and will do redundant content pushes.

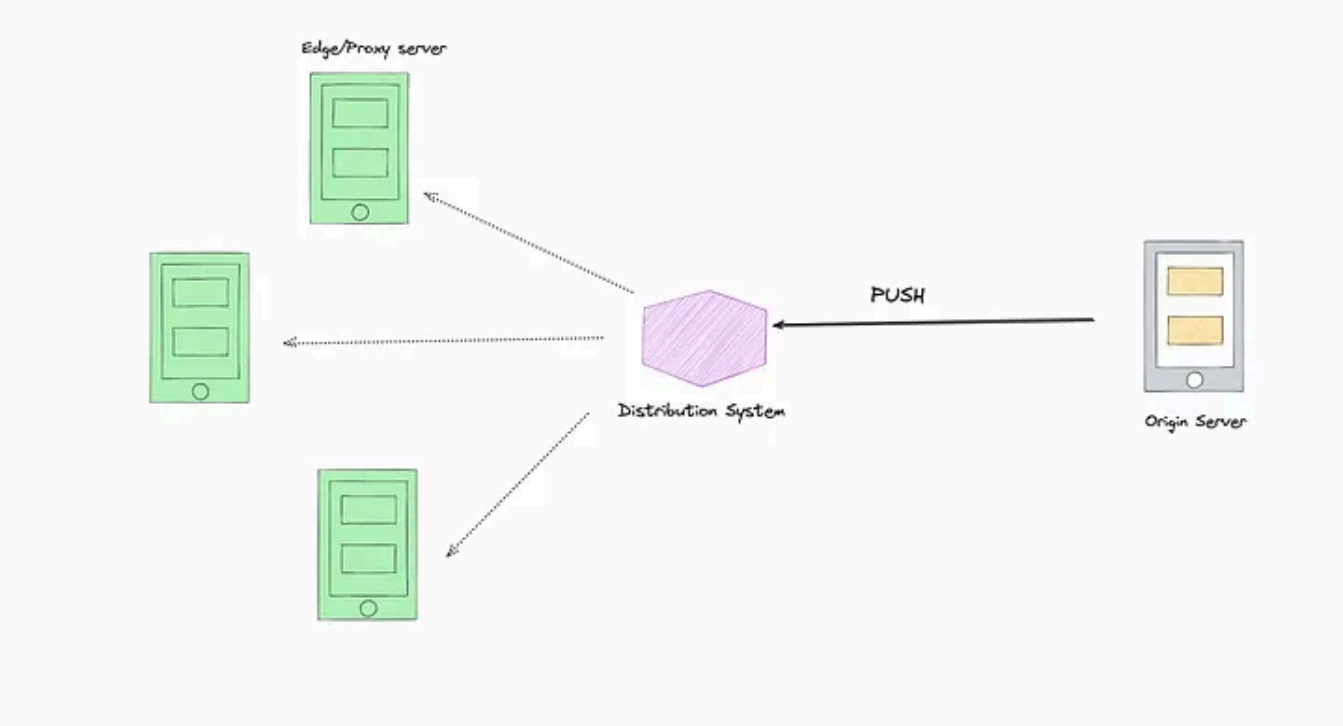

Fig 2.0: PUSH CDN

Pull CDN

A CDN **pulls** the unavailable data from origin servers when requested by a user. The proxy servers keep the files for a specified amount of time and then remove them

from the cache if they're no longer requested to balance capacity and cost.

When users request web content in the pull CDN model, the CDN itself is responsible for pulling the requested content from the origin server and serving it to the users. Therefore, this type of CDN is more suited for serving **dynamic content**.

Since static content is served to a wide range of users for longer than dynamic content, the push CDN scheme maintains **more replicas** than the pull CDN, thus improving availability. On the other hand, the pull CDN is favored for frequently changing content and a high traffic load. L**ow storage consumption** is one of the main benefits of the pull CDN.

Dynamic content caching optimisation

Since dynamic content often changes, it's a good idea to cache it optimally. It's **optimal to run the scripts at proxy servers instead of the origin servers**.

Certain dynamic content creation requires the execution of scripts that can be executed at proxy servers instead of running on the origin server. Dynamic data can be generated using various parameters, which can be beneficial if executed at the proxy servers. For example, we can generate dynamic content based on user location, time of day at a location, third-party APIs specific to a location (for instance, weather API), and so on.

To reduce the communication between the origin server and proxy servers and storage requirements at proxy servers, it's useful to employ compression techniques as well

Multi-tier CDN architecture

The content provider sends the content to a **large number** of clients through a CDN. The task of distributing data to all the CDN proxy servers simultaneously is challenging and **burdens the origin server** significantly. CDNs follow a **tree-like structure** to ease the data distribution process for the origin server. The edge proxy servers have some peer servers that belong to the same hierarchy. This set of servers receives data from the parent nodes in the tree, which eventually receive data from the origin servers. The data is copied from the origin server to the proxy servers by following different paths in the tree.

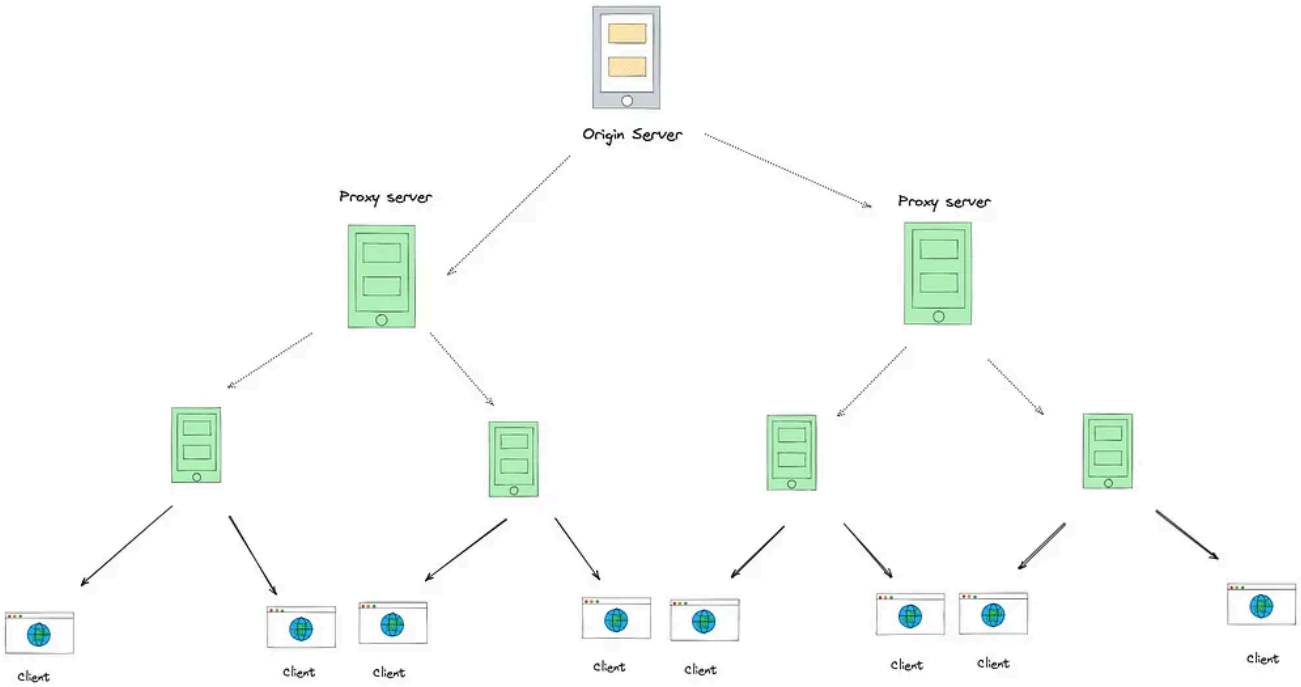

Fig 3.0: Multi-tier CDN archicture

Advantages:

- Scale our system for increasing users by adding more server nodes to the tree.
- It also reduces the burden on the origin server for data distribution.

Here, a **multi-layer cache** might be used to handle long-tail of less popular content.

Content consistency in CDN

Data in the **proxy servers** should be **consistent** with data in the **origin servers**. There's always a risk of users accessing stale data if the proxy servers don't remain consistent with the origin servers. Different consistency mechanisms can be used to ensure consistency of data, depending on the push or pull model.

Periodic polling

Using the **pull** model, proxy servers **request the origin server periodically** for updated data and change the content in the cache accordingly. When content changes infrequently, the polling approach consumes unnecessary bandwidth. Periodic polling uses **time-to-refresh (TTR)** to adjust the time period for requesting updated data from the origin servers.

Time-to-live (TTL)

Because of the TTR, the proxy servers may uselessly request the origin servers for updated data. A better approach that could be employed to reduce the frequency of refresh messages is the **time-to-live (TTL)** approach. In this approach, each object has a TTL attribute assigned to it by the origin server. The TTL defines the **expiration time of the content**. The proxy servers serve the same data version to the users until that content expires. Upon expiration, the proxy server checks for an update with the origin server. If the data is changed, it gets the updated data from the origin server and then responds to the user's requests with the updated data. Otherwise, it keeps the same data with an updated expiration time from the origin servers.

Leases

The origin server grants a lease to the data sent to a proxy server using this technique. The **lease** denotes the time interval for which the origin server agrees to notify the proxy server if there's any change in the data. The proxy server must send a message requesting a lease renewal after the expiration of the lease. The lease method helps to reduce the number of messages exchanged between the proxy and origin server. Additionally, the lease duration can be optimised dynamically according to the observed load on the proxy servers. This technique is referred to as an **adaptive lease**.

Evaluation

Performance

CDN achieves high performance by **minimising latency**. Some of the key design decisions that minimise latency are as follows:

- Proxy servers usually serve content from the **RAM**.
- CDN proxy servers are **placed near the users** to provide faster access to content.
- The proxy servers have long-tail content stored in nonvolatile storage systems like SSD or HDD. Serving from these resources results in a more negligible latency than we'd see from serving content from origin servers.
- As was discussed previously, proxy servers can be implemented in layers where if one layer doesn't have the content, the request can be entertained by the next layer of proxy servers. For example, the edge proxy servers can request the parent proxy servers.

Availability

A CDN can deal with massive traffic due to its distributed nature. A CDN ensures **availability through its cached content** that serves as a backup whenever the origin servers fail. Moreover, if one or more proxy servers in the CDN stop working, other operational proxy servers step in and continue to drive the web traffic. In addition, edge proxy servers can be made available through **redundancy by replicating data** to as many proxy servers as needed to avoid a single point of failure and to meet the request load.

Scalability

The design of CDN facilitates scalability in the following ways:

- It brings content closer to the user and removes the requirement of high bandwidth, thereby ensuring scalability.
- Horizontal scalability is possible by adding the number of reading replicas in the form of edge proxy servers.
- The limitations with horizontal scalability and storage capacity of an individual proxy server can be dealt with using the layered architecture of the proxy servers we described above.

CDN setup

Setting up CDN involves several steps includes — Create a Load Balancer, Configure the CDN, and Update DNS Records.

Content Delivery Network | Cdn Providers | Software Architecture | Software Architect | Distributed Systems

Edit profile

Written by Suresh Podeti